Obiettivi E Strumenti

Sommario

Lo scopo di questo progetto è quello di eseguire una dettagliata analisi di immagini attraverso l'uso delle tecniche del Deep Learning. In particolare si affronta qui il problema della Classificazione, ovvero quello di identificare fra un insieme di classi quella a cui appartiene un dato input (*object recognition*). Lo sviluppo segue i dettami delle reti neurali e del Supervised Learning: viene allenata la rete attraverso un training set composto da immagini di input e corrispondenti label, cosicché questa apprenda il task della classificazione in modo sempre più accurato con il succedersi delle iterazioni. L'obiettivo è sviluppare un modello performante su diversi dataset, confrontando i risultati ottenuti e perfezionandoli attraverso la modifica del modello stesso e dei suoi (iper)parametri.

Seguirà nei prossimi paragrafi una breve panoramica sulle astrazioni e gli strumenti utilizzati per il perseguimento di quanto precedentemente dichiarato. Le reti neurali[1] ricoprono un ruolo di primo ordine nel corretto funzionamento dell'applicazione e, soprattutto, nel raggiungimento di risultati che altrimenti non sarebbero mai stati ottenuti. Per questo motivo si ritiene necessario indagare, anche se in una trattazione poco esaustiva, la loro struttura e le loro caratteristiche, ponendo in evidenza le scelte che sono state adottate nei programmi e le motivazioni che implicano quelle scelte.

Neuroni Ed Attivazione

Costituente imprescindibile di ogni rete neurale, sia essa totalmente connessa o di convoluzione, è certamente il neurone, unità computazionale deputata alla trasmissione dell'informazione dagli strati precedenti agli strati successivi attraverso l'uso di una funzione di attivazione. Lo schema è perfettamente riassunto nella figura 1.

La funzione di attivazione è un elemento fondamentale, utile per aumentare la complessità del modello e renderlo così in grado di eseguire compiti

[1] Nel contesto dell'attuale progetto si ricorrerà all'uso di reti totalmente connesse e di reti convoluzionali, le più adatte ed utilizzate agli scopi prefissati

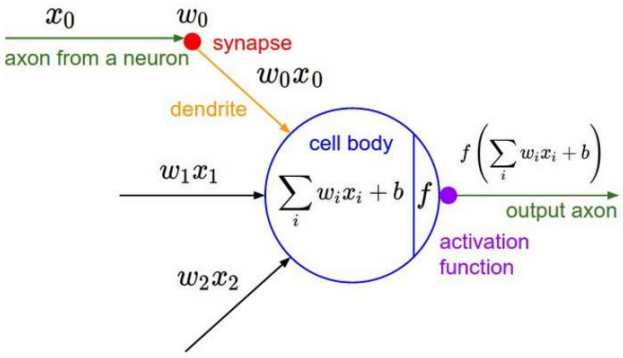

Figura 1: Rappresentazione della struttura di un neurone

decisamente più articolati. In assenza di una opportuna trasformazione la rete potrebbe infatti eseguire solo funzioni lineari (attraverso l'uso dei pesi e dei bias associati ai diversi strati). Di seguito sono elencate alcune delle più popolari funzioni di attivazione

Funzione Logistica	$f(x) = \dfrac{1}{1+e^{-x}}$
Arcotangente	$f(x) = tan^{-1}(x)$
Tangente Iperbolica	$f(x) = \dfrac{2}{1+e^{-2x}} - 1$
ReLU (Rectified Linear Units)	$f(x) = max(0, x)$

Le funzioni presentate hanno output diversi e si prestano quindi ad essere utilizzate in contesti differenti, a seconda del problema in questione. Tutte hanno però una caratteristica comune: sono differenziabili. Questo diventa un elemento imprescindibile nel momento in cui si esegue la backpropagation, ovvero il processo attraverso cui si allena la rete.

Per il progetto è stata scelta la funzione ReLU, la più popolare negli ultimi anni perché in grado di rimpiazzare le precedenti superando problemi come il cosiddetto *"Vanishing Gradient Problem"*. È necessario osservare che, ad oggi, si utilizzano anche diverse varianti, come la funzione Leaky ReLU, per cotrastare nuove difficoltà quali il *"Dying Neuron Problem"*.

Lo strato di output non può fare ricorso alla funzione ReLU e si ricorre quindi, nel caso presente della classificazione, alla funzione Softmax

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \qquad \text{con } j = 1, \ldots, K$$

Errore Ed Ottimizzazione

L'output ottenuto dalla rete, a seguito dell'esecuzione della funzione Softmax, può essere interpretato come una distribuzione di probabilità che indica l'appartenenza degli input alle diverse classi. In questa situazione, in alternativa al più celebre Errore Quadratico Medio, si ricorre ad una diversa funzione di costo: la *Cross Entropy*. Detti rispettivamente y_k e p_k la probabilità reale e la probabilità predetta dalla rete che un dato oggetto appartenga alla classe k-esima:

$$H(\mathbf{y}, \mathbf{p}) = -\sum_{k=1}^{K} y_k \, log(p_k)$$

L'obiettivo diventa quindi quello di minimizzare la funzione di costo attraverso un determinato metodo di ottimizzazione. Il più celebre di questi metodi è quello del *Gradient Descent*, procedimento che fa uso delle derivate parziali per procedere nella direzione verso cui la funzione decresce (figura 2). In altre parole, il gradiente, rappresentato da un matrice Jacobiana, è impiegato per aggiornare i pesi della rete nel già citato procedimento di backpropagation.

$$\theta = \theta - \eta \, \nabla J(\theta)$$

In alcuni casi si ricorre alla variante dello *Stochastic Gradient Descent* per superare alcuni limiti come la convergenza in un minimo locale della funzione di costo. In ogni caso, è comunque importante settare il learning rate ad un valore che sia sufficientemente equilibrato.

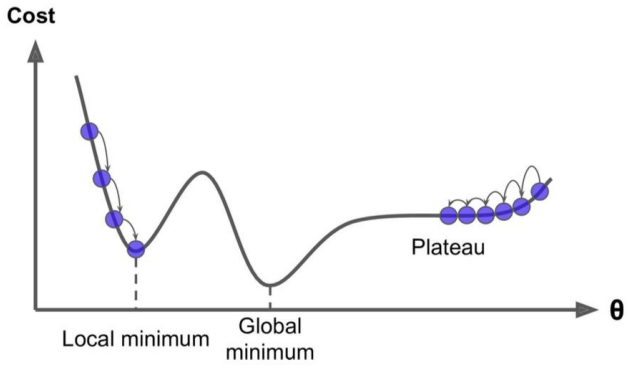

Figura 2: Ottimizzare la funzione di costo: discesa del gradiente

Nel corso dei successivi moduli vengono utilizzate anche altre tecniche per l'ottimizzazione, più complesse dal punto di vista matematico-teorico ma utili per ottenere risultati significativi preclusi al Gradient Descent. Non se ne approfondiranno quindi gli aspetti più tecnici che, pur essenziali per una visione completa, esulano dagli obiettivi del progetto.

Convoluzione

Elemento centrale del progetto sono senza dubbio le reti neurali di convoluzione: queste sono le più adeguate nell'elaborazione delle immagini di input e, in particolare, nella risoluzione del problema della Classificazione.

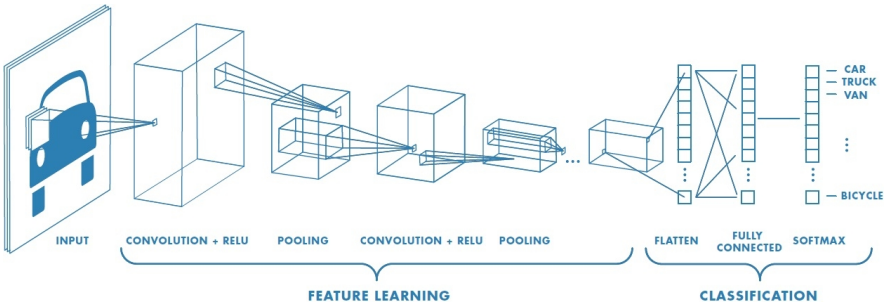

Figura 3: Rete neurale convoluzionale

Le reti convoluzionali utilizzate nello sviluppo del codice si articolano nei seguenti strati:

kernel filtro che esegue la vera e propria operazione di convoluzione sulla matrice di input. Nel progetto si fa uso della sola convoluzione a due dimensioni. Se si utilizza un kernel di dimensioni W_K e H_K, un padding P e uno stride (S_w, S_h), la dimensione delle matrici convolute è data dalle seguenti relazioni

$$W = \frac{W - W_K + 2P}{S_w} + 1 \qquad H = \frac{H - H_K + 2P}{S_h} + 1$$

relu funzione di attivazione descritta in precedenza

pooling operazione utile per il ridimensionamento delle matrici. Nel progetto si fa uso della forma di pooling più comune: il max-pooling (figura 4).

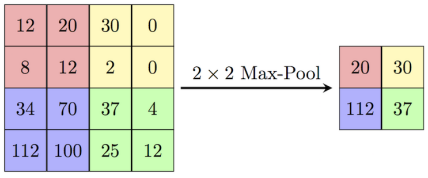

Figura 4: Uso del max-pooling in una matrice d'esempio

Nella parte finale della rete, prima di applicare il Softmax ed ottenere così la predizione elaborata dalla stessa, è necessario ricorrere ad uno o più strati totalmente connessi. Per far ciò si intoduce un flatten layer che, come indica il nome stesso, consente una ristrutturazione della matrice (o del tensore) ottenuta dagli strati di convoluzione.

Risultati

Come anticipato, il progetto ha molteplici obiettivi: non solo giungere ad un risultato utile alla classificazione delle immagini fornite come input, ma anche analizzare alcune delle principali tecniche del Deep Learning e confrontarne le prestazioni. A tal fine sono stati implementati cinque programmi autoconsistenti, i cui risultati sono illustrati nei paragrafi che seguono.

Modulo 1

Il primo modulo ha la funzione di introdurre il problema e testare una modalità di risoluzione essenziale che, come apparirà evidente, riuscirà ad ottenere ottimi risultati dipendentemente dal dataset prescelto. In particolare è stato utilizzata una rete costituita da un singolo strato totalmente connesso e caratterizzata dai già discussi metodi di Softmax, Cross-Entropy e Gradient Descent.

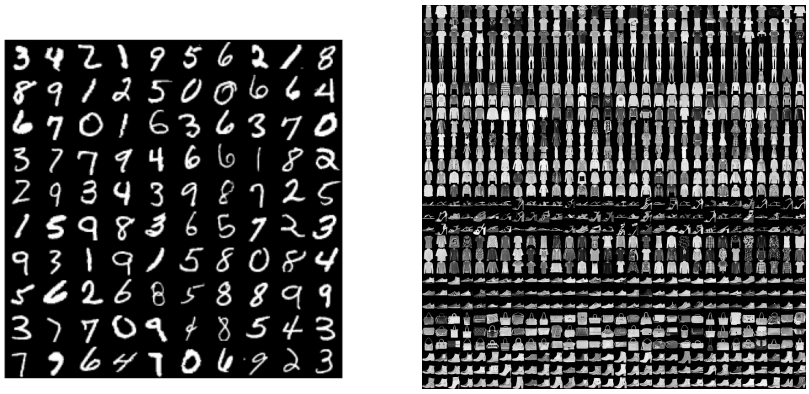

Figura 5: Immagini dai dataset MNIST e Fashion-MNIST

L'interesse è ricaduto nell'utilizzo di differenti dataset. Il primo e più celebre e certamente MNIST[2], un insieme di immagini rappresentanti le cifre del sistema decimale scritte a mano. Questo è senza dubbio il più largamente

[2]I link per reperire tutte le risorse utilizzate sono disponibili nella bibliografia al termine di questo documento

utilizzato e sfruttato, e quindi anche il più adatto per introdurre l'analisi e la classificazione. Entrando nel merito, le immagini utilizzate sono a scala di grigi e di dimensione 28x28. Il training set si compone di 55000 esempi, mentre il test set (utilizzato per verificare la bontà nell'apprendimento della rete) ne presenta 10000.

Al precedente dataset ne è stato affiancato un secondo, Fashion-MNIST. Come suggerisce lo stesso nome, le caratteristiche rimangono del tutto invariate ad eccezione del contenuto delle immagini, che rappresentano questa volta un insieme di capi di abbigliamento. Le dieci categorie in cui si dividono gli input (fra cui ci sono "T-shirt", "Trouser" e "Pullover") vengono stampate a schermo durante l'esecuzione del programma.

Nel processo di training è stato diviso il numero di immagini in mini-batch, ed è stata allenata la rete utilizzato tutti questi sottoinsiemi per ogni epoca. I risultati ottenuti sono riassunti nella seguente tabella.

Dataset Scelto	Learning Rate	Numero Epoche	Dimensione Batch	Training Accuracy	Test Accuracy
MNIST	0.01	35	100	95.47%	91.99%
MNIST	0.001	35	100	93.16%	92.43%
MNIST	0.001	35	5000	82.18%	81.98%
Fashion-MNIST	0.01	35	100	85.86%	78.68%
Fashion-MNIST	0.001	35	100	87.79%	83.97%
Fashion-MNIST	0.001	35	5000	69.68%	68.99%

Durante l'esecuzione del programma è possibile scegliere il dataset da utilizzare e se visualizzare una immagine d'esempio. I parametri del modello si possono invece modificare intervenendo direttamente nel codice sorgente.

Modulo 2

Nel secondo modulo si vuole osserevare la variazione di performance ottenute utilizzando diversi metodi di ottimizzazione. Nei precedenti paragrafi si è parlato del Gradient Descent, ma si può procedere oltre e migliorare ulteriormente la tecnica per minimizzare il costo ed aggiornare i pesi della rete. Alcuni di questi sono[3]:

Momentum, che perfeziona l'instabile ed oscillatoria convergenza dello Stochastic Gradient Descent, accelerando l'andamento lungo le direzioni più rilevanti e decellerandolo lungo le meno rilevanti

$$V(t) = \gamma V(t-1) + \eta \nabla J(\theta_t) \quad \Rightarrow \quad \theta_{t+1} = \theta_t - V(t)$$

[3]I metodi introdotti costituiscono una rapida panoramica senza alcuna pretesa di esaustività nella presentazione, funzionale al solo utilizzo dei suddetti metodi all'interno del presente progetto

Nesterov Accelerated Gradient, che supera il problema del possibile mancato arresto negli aggiornamenti dei parametri θ una volta raggiunto il minimo della funzione di costo

$$V(t) = \gamma V(t-1) + \eta \nabla J(\theta_t - \gamma V(t-1)) \quad \Rightarrow \quad \theta_{t+1} = \theta_t - V(t)$$

AdaGrad, che permette al learning rate di variare ed adattarsi automaticamente in funzione del parametro che si cerca di aggiornare, diminuendo progressivamente il valore del learning rate stesso in base al gradiente calcolato in precedenza

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} g_{t,i}$$

AdaDelta, che supera il problema della progessiva diminuzione del learning rate, probabile causa nel caso di AdaGrad di una prematura interruzione nell'apprendimento

$$E[g^2](t) = \gamma E[g^2](t-1) + (1-\gamma) g^2(t) \quad \Rightarrow \quad \Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Adam, che coniuga i vantaggi di Momentum e AdaDelta

$$\hat{m}_{t+1} = \frac{\gamma_1 m_t + (1-\gamma_1)\nabla J(\theta_t)}{1-\gamma_1^{t+1}} \quad \wedge \quad \hat{g}_{t+1} = \frac{\gamma_2 g_t + (1-\gamma_2)\nabla J(\theta_t)^2}{1-\gamma_2^{t+1}}$$

$$\Rightarrow \ \theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{g}_t} + \epsilon} \hat{m}_{ts}$$

Per l'utilizzo ed il confronto dei metodi di ottimizzazione si utilizza un nuovo dataset, "Quick, Draw!", le cui immagini 28x28 afferiscono a numerosissime categorie di disegni fatti a mano.

Per poter apprezzare le variazioni di performance senza introdurre elementi di distorsione, non è stata apportata alcuna modifica alla rete neurale totalmente connessa usata in precedenza. Si ha quindi ancora una volta un modello decisamente semplice ma altrettanto efficace per gli scopi prefissati. Si può notare invece una differenza nel processo di training, che questa volta non utilizza la divisione in batch del training set.

Dataset Scelto	Learning Rate	Numero Epoche	Optimizer Scelto	Training Accuracy	Test Accuracy
Quick, Draw!	0.001	100	GD	63.83%	63.73%
Quick, Draw!	0.001	100	AdaGrad	66.22%	66.18%
Quick, Draw!	0.001	100	Adam	73.73%	73.73%
Quick, Draw!	0.01	100	GD	59.75%	59.59%
Quick, Draw!	0.01	100	AdaGrad	76.01%	76.22%
Quick, Draw!	0.01	100	Adam	79.11%	79.24%

Modulo 3

Il terzo modulo è deputato all'introduzione delle reti convoluzionali. I dataset utilizzati sono nuovamente MNIST e Fashion-MNIST ma questa volta è il modello che cambia la propria natura: uno o due strati di convoluzione seguiti da uno o due successivi strati totalmente connessi, che conducono al termine con la funzione Softmax.

È interessante notare come le operazioni di ReLU e max-pooling possono essere disposte l'una prima dell'altra indipendentemente in ogni strato di convoluzione, ovvero

$$MaxPool(ReLU(Conv(M))) = ReLU(MaxPool(Conv(M)))$$

Questa proprietà è verificata perchè la funzione ReLU opera elemento per elemento ed è non decrescente, ma non vale in generale per ogni funzione di attivazione ed ogni metodo di pooling. In questa occasione si pospone la funzione ReLU in modo da diminuire il numero di operazioni da eseguire (anche se questo non modifica la velocità di esecuzione del programma). Inoltre, in questo modulo come nei successivi, si affianca all'uso del Gradient Descent quello di Adam (Adaptive Moment Estimation), che emerge come il migliore dei metodi di ottimizzazione nel precedente test delle prestazioni.

Utilizzando uno strato di convoluzione con 16 filtri di dimensione 5x5 e un solo strato totalmente connesso con 10 neuroni terminali (pari al numero di categorie), si ottengono i seguenti risultati.

Dataset Scelto	Learning Rate	Numero Epoche	Optimizer Scelto	Training Accuracy	Test Accuracy
MNIST	0.001	20	GD	90.31%	91.21%
MNIST	0.001	5	Adam	98.31%	98.03%
Fashion-MNIST	0.001	20	GD	81.21%	80.04%
Fashion-MNIST	0.001	5	Adam	89.34%	88.24%

Aggiungendo un nuovo strato di convoluzione con 32 filtri e uno strato totalmente connesso con 128 neuroni (la struttura della rete presente nel programma), si ha:

Dataset Scelto	Learning Rate	Numero Epoche	Optimizer Scelto	Training Accuracy	Test Accuracy
MNIST	0.001	20	GD	92.38%	92.74%
MNIST	0.001	5	Adam	99.45%	98.94%
MNIST	0.01	5	Adam	99.53%	98.47%
Fashion-MNIST	0.001	20	GD	78.98%	76.59%
Fashion-MNIST	0.001	5	Adam	91.30%	89.48%
Fashion-MNIST	0.01	5	Adam	92.17%	88.28%

Modulo 4

Il quarto modulo riassume tutto ciò che è stato descritto e inferito empiricamente in precedenza, applicando le tecniche di apprendimento ad un nuovo dataset: German Traffic Signs. Come indica il nome stesso, le immagini rappresentano numerosi segnali stradali, divise in 43 categorie diverse e di dimensioni 32x32.

Il modello di rete convoluzionale è simile al precedente e per questo non è necessario soffermarsi sulle caratteristiche. Il percorso nello studio del Deep Learning e del problema della Classificazione si sofferma questa volta sull'utilizzo del validation set, un insieme di input che affianca il training set ed il test set. In particolare si utilizza il validation set per valutare la bontà dell'apprendimento eseguito dalla rete durante la fase di allenamento e settare al meglio i parametri del modello.

Figura 6: Immagini dal dataset German Traffic Signs

Sfruttando il validation set si possono analizzare i celebri problemi di overfitting e underfitting, eventualmente osservando i grafici delle cosiddette "learning curves". Se la rete ottiene scarsi risultati tanto nel training set quanto nel validation set allora si ha una probabile situazione di underfitting. Al contrario, se la rete ha ottime performance sul training set ma non altrettanto sul validation set, allora si è in presenza di overfitting.

Per quanto riguarda il caso particolare del progetto, si è cercato proprio di fare questo: utilizzare il validation set come strumento di rilevazione per le possibili distorsioni nell'apprendimento, oltre che come ulteriore insieme di input per testare i risultati.

Dataset Scelto	Learning Rate	Numero Epoche	Training Accuracy	Validation Accuracy	Test Accuracy
Traffic Signs	0.01	10	96.86%	85.12%	84.39%
Traffic Signs	0.001	5	98.60%	89.30%	89.63%
Traffic Signs	0.001	10	99.60%	88.64%	91.16%
Traffic Signs	0.001	20	99.98%	94.06%	92.78%
Traffic Signs	0.0001	20	97.59%	82.97%	84.18%

Modulo 5

Nel quinto ed ultimo modulo si chiude l'analisi della Classificazione con ulteriori criteri di raffinazione della rete e due nuovi dataset. Il primo è SVHN (Street View House Numbers), simile a MNIST nel rappresentare le dieci cifre del sistema numerico ma decisamente più significativo: si tratta infatti di un insieme di numeri civici che evidenziano come lo sviluppo della rete una diretta applicazione nel mondo reale. Il secondo è invece CIFAR-10, un insieme di input costituito da immagini di dimensione 32x32 a colori (caratterizzate quindi dai tre canali canonici Red, Green, Blue).

Figura 7: Immagini dai dataset SVHN e CIFAR-10

Per migliorare ulteriormente le prestazioni della rete neurale ed ottenere nuovi risultati per la risoluzione del problema in analisi si può ricorrere ai metodi di regolarizzazione, in grado di evitare eventuali overfitting dei dati da parte del modello. Ai dataset presentati sono state applicate sostanzialmente due regolarizzazioni: l2 e dropout.

La regolarizzazione l2 consiste nell'inserire un termine aggiuntivo alla funzione costo, del tipo

$$R(\beta) = \sum_{j=1}^{p} \beta_j^2$$

La nuova funzione di costo diventa quindi simile alla seguente

$$E = H(y, p) + \lambda R(\beta)$$

Se λ è uguale a zero vuol dire che non si applica nessuna regolarizzazione e il costo rimane invariato. Se, al contrario, lambda fosse troppo elevato, si rischia di riscontrare il problema di underfitting. L'obiettivo è penalizzare i parametri con valori troppi elevati nel corso del loro aggiornamento.

Dropout è un'altra tecnica ampiamente utilizzata per la regolarizzazione ma di diversa natura. In questo caso, ad ogni passo del processo di allenamento, si ha una certa probabilità d che una determinata unità della rete sia

effettivamente attiva e partecipi nel suo ruolo, e una probabilità $1 - d$ che l'unità venga invece ignorata durante il forward e la backpropagation.

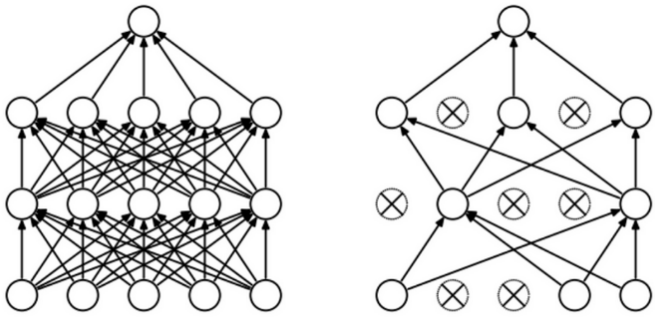

Figura 8: Schema di una rete in cui si applica il dropout

Con 5 epoche ed un learning rate pari a 0.001, i risultati ottenuti sono riassunti in questa ultima tabella

Dataset Scelto	Tipo Regol.	Parametro Relativo	Training Accuracy	Validation Accuracy	Test Accuracy
SVHN	-	-	95.26%	87.75%	90.57%
SVHN	L2	$l = 0.01$	91.80%	88.13%	90.95%
SVHN	dropout	$d = 0.5$	92.82%	88.78%	91.64%
CIFAR-10	-	-	75.68%	70.02%	70.68%
CIFAR-10	L2	$l = 0.01$	64.76%	64.43%	64.22%
CIFAR-10	dropout	$d = 0.5$	71.39%	68.72%	68.69%

Risorse Utilizzate

Libri

ML Hands-On Machine Learning, Aurélien Géron, O'Reilly

Dataset

MNIST http://yann.lecun.com/exdb/mnist/

Fashion-MNIST https://github.com/zalandoresearch/

Quick, Draw! https://quickdraw.withgoogle.com/data

Traffic Signs http://benchmark.ini.rub.de/

SVHN http://ufldl.stanford.edu/housenumbers/

CIFAR-10 https://www.cs.toronto.edu/~kriz/cifar.html

Immagini

Neurone stackoverflow.com/single-neuron

CNN mathworks.com/convolutional-neural-network

Pooling computersciencewiki.org/max-pooling

MNIST knowm.org/mnist

Traffic Signs researchgate.net/ftraffic-sign-classes

Approfondimenti

Ottimizzazione towardsdatascience.com/optimization-algorithms

Dropout medium.com/dropout-in-deep-machine-learning

Regolarizzazione towardsdatascience.com/l2-regularization